Available online at www.sciencedirect.com
SCIENCE DIRECT
Applied Mathematics and Computation 178 (2006) 103–117
APPLIED MATHEMATICS AND COMPUTATION
www.elsevier.com/locate/amc

Analog computation beyond the Turing limit

Jerzy Mycka

Institute of Mathematics, University of Maria Curie-Sklodowska, pl M. Curie-Sklodowskiej 1, 20-031 Lublin, Poland

Abstract

The main purpose of this paper is quite uncontroversial. First, we recall some models of analog computations (including these allowed to perform Turing uncomputable tasks). Second, we support the suggestions that such hypercomputable capabilities of such systems can be explained by the use of infinite limits. Additionally, the inner restrictions of analog models of computations are indicated.

1. Preliminaries

Alan Turing clarified the notion of algorithm giving it a precise meaning as well as introduced a coherent framework for discrete computation. Soon, new results showing the relations of his model with other approaches, such as recursive functions (in the sense of Kleene) or Church's λ-calculus (for information about this subject, see Odifreddi [21]), originated in a natural way consistent theoretical basis to standard computation theory. However, all these models use enumerable domains and treat time of computations as discrete.

Nevertheless, computers need not have such qualities. Analog computers with the continuous internal states rather than discrete, as in digital computation, were invented and discussed quite thoroughly. Unfortunately, because of the problem of a coherent theoretical basis for analog computation and the fact that analog computers technology improved insignificantly in the second half of the last century, when compared with its digital counterpart, analog computation was about to be forgotten. For many reasons (new paradigms of computation, search for good tools for numerical analysis, new technologies) this situation seems to change.

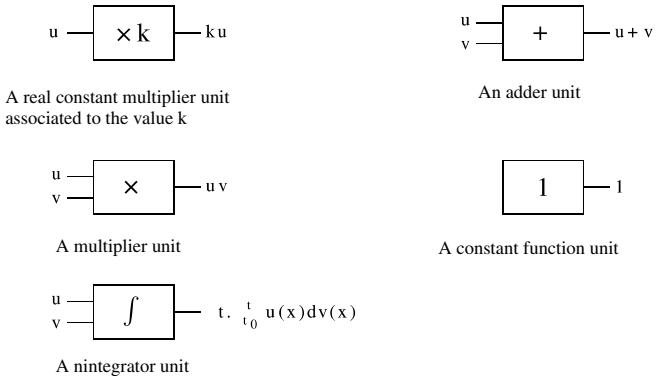
We may classify analog models as discrete time models (e.g., [1]) or as continuous time models. In this paper we are interested in the latter type. The basic model in this field is Shannon's General Purpose Analog Computer [29]. The General Purpose Analog Computer (GPAC) is a computer whose computation evolves in continuous time. The outputs are generated from the inputs by means of dependences defined by a finite directed graph (not necessarily acyclic) where each node is one of the following boxes.

- *Integrator*: a two-input, one-output unit with a setting for initial condition. If the inputs are unary functions u, v, then the output is the Riemann–Stieltjes integral $\lambda t \cdot \int_{t_0}^{t} u(x)\,\mathrm{d}v(x) + a$, where a and t_0 are real constants defined by the initial settings of the integrator.

E-mail address: Jerzy.Mycka@umcs.lublin.pl

0096-3003/$ - see front matter
doi:10.1016/j.amc.2005.09.074

- *Constant multiplier*: a one-input, one-output unit associated to a real number. If u is the input of a constant multiplier associated to the real number k, then the output is ku.
- *Adder*: a two-input, one-output unit. If u and v are the inputs, then the output is $u+v$.
- *Multiplier*: a two-input, one-output unit. If u and v are the inputs, then the output is uv.
- *Constant function*: a zero-input, one-output unit. The value of the output is always 1.



Representations of different types of units in a GPAC.

Although the above notion of GPAC seems fairly intuitive and natural, the accepted definition is due to Pour-El (see [23]). Let us now present a precise version of her definition for functions of one variable. In the following, I will denote a closed bounded interval with non-empty interior.

Definition 1 [23]. The unary function y is generated by a GPAC on I if there exist a set of unary functions $y_1,\ldots,y_n$ and a set of initial conditions $y_i(a)=y_i^*,\ i=1,\ldots,n$, where $a\in I$, such that,

1. $\bar{y}=(y_1,\ldots,y_n)$ is the unique solution on I of a system of ODEs of the form
$$A(x,\bar{y})\frac{\mathrm{d}\bar{y}}{\mathrm{d}x}=b(x,\bar{y}) \tag{1}$$
satisfying the initial conditions, where $A(x,\bar{y})$ and $b(x,\bar{y})$ are $n\times n$ and $n\times 1$ matrices, respectively. Furthermore, each entry of A and b must be linear in $1,x,y_1,\ldots,y_n$.
2. For some $1\leqslant i\leqslant n$, $y=y_i$ on I.
3. $(a,y_1^*,\ldots,y_n^*)$ has a domain of generation with respect to the above equation, i.e., there are closed intervals $J_0,J_1,\ldots,J_n$ (with non-empty interiors) such that $(a,y_1^*,\ldots,y_n^*)$ is an interior point of $J_0\times J_1\times\cdots\times J_n$ and, furthermore, whenever $(b,z_1^*,\ldots,z_n^*)\in J_0\times J_1\times\cdots\times J_n$, there exist unary functions $z_1,\ldots,z_n$ such that,
   (i) $z_i(b)=z_i^*$ for $i=1,\ldots,n$;
   (ii) $(z_1,\ldots,z_n)$ satisfy Eq. (1) on some interval I^* with non-empty interior such that $b\in I^*$;
   (iii) $(z_1,\ldots,z_n)$ is unique on I^*.

The existence of a domain of generation indicates that the solution of the above equation remains unique for sufficiently small changes on the initial conditions.

Let us recall that a function $f(x)$ is differentially algebraic [25] if its derivatives satisfy a polynomial equation $P(x,f(x),\ldots,f^{(k)}(x))=0$ for some polynomial with rational coefficients. A function of several variables is differentially algebraic if it is a differentially algebraic function of each variable when the others are fixed. Provided with the above definition, Pour-El shows the following result.

Theorem 2 [23]. *If y is generable on I by a GPAC, then there is a closed subinterval $I'\subseteq I$ with non-empty interior such that on I', y is differentially algebraic.*

Another important model of analog computation is Rubel's Extended Analog Computer (EAC) [27]. This model is similar to the GPAC, but we allow other types of units and several independent variables because Rubel does not seek any equivalence with existing models. The EAC permits all the operations of ordinary analysis, except the unrestricted taking of limits. The new units add an extended computational power relatively to the GPAC.

The EAC works on a hierarchy of levels, getting more versatile as one goes to higher levels in the hierarchy. At the lowest level 0, it produces and manipulates real polynomials of any finite number of real variables. The EAC has no inputs from outside, but it has a finite number of settings (arbitrary real numbers). The settings determine behavior of the machine from the first level through all subsequent levels. The EAC has outputs and we demand them to be real analytic. The outputs at level $n-1$ can be used as inputs at level n. The EAC satisfied the condition that when inputs at some level are modified by small errors then the outputs differ from the original output only by a small amount on each compact set.

At each level, the actual computing is done by some boxes of different kinds. First, there are the *constant* boxes, which produce arbitrary real constants. There are *projection* boxes, which produce any of variables $x_1,\ldots,x_n$. Later, there are *adders*, *multipliers* and *substituters* (for given $v,u_1,\ldots,u_k$ they give $v(u_1,\ldots,u_k)$ as the output). Moreover, there are *inverters*, which for given $f_1,\ldots,f_k:\mathbb{R}^{n+k}\to\mathbb{R}$ find real analytic functions $y_1,\ldots,y_k$ of $x_1,\ldots,x_n$ such that for all $1\leqslant i\leqslant k$ we have $f_i(x_1,\ldots,x_n,y_1,\ldots,y_k)=0$. We have *differentiators*, that for any function produce desirable mixed derivatives. Certain sets in Euclidean space are produced by the machine for a given function f, namely $\{(x_1,\ldots,x_n):f(x_1,\ldots,x_n)>0\}$ or $\{(x_1,\ldots,x_n):f(x_1,\ldots,x_n)\geqslant 0\}$. Then there are the *analytic continuation* boxes, which produce a unique analytic continuation for given function and domain.

The quintessential box is the *boundary-value problem* box, which solves a finite set of partial and ordinal differential equations on the given set with certain bound and boundary values prescribed. For some given function f and set Ω with the given boundary we can also use *restricted limit* box, which defines $\phi(x)=\lim_{y\to x,y\in\Omega}f(y)$.

Using the results from [25,27] we can formulate the following statement.

Proposition 3 [27]. *The set of GPAC-computable functions is a proper subset of the set of EAC-computable functions.*

For example, the EAC can generate the Γ- and ζ-functions (it is known that the GPAC cannot solve these problems [25]). It is not known if a physical version of the EAC exists.

A new approach was given by Moore in 1996. In the work [13], he defined a set of (vector-valued) functions on the reals (called $\mathbb{R}$-recursive functions) in the analogous way to the classical recursive functions on the natural numbers. His model has also a continuous time of computation (a continuous integration instead of a discrete recursion). Moore's seminal paper gave rise to a further development in real recursive function theory (in spite of some problems with his definition and results).

In [18] one can find the definition, which is a derivative of Moore's original formulation, introduced to avoid problems involved in the latter. It is important to see that the following definition is based on the vector operations.

Definition 4 [18]. The set of real recursive vectors is generated from the real recursive scalars 0, 1, -1 and the real recursive projections $I_n^i(x_1,\ldots,x_n)=x_i,\ 1\leqslant i\leqslant n,\ n>0$, by the operators:

1. *Composition*: if f is a real recursive vector with n k-ary components and g is a real recursive vector with k m-ary components, then the vector with n m-ary components ($1\leqslant i\leqslant n$)

   $$\lambda x_1\ldots x_m\cdot f_i(g_1(x_1,\ldots,x_m),\ldots,g_k(x_1,\ldots,x_m))$$

   is real recursive.
2. *Differential recursion*: if f is a real recursive vector with n k-ary components and g is a real recursive vector with n $(k+n+1)$-ary components, then the vector h of n $(k+1)$-ary components, which is the solution of the Cauchy problem for $1\leqslant i\leqslant n$

   $$h_i(x_1,\ldots,x_k,0)=f_i(x_1,\ldots,x_k),$$

$$\partial_y h_i(x_1,\ldots,x_k,y) = g_i(x_1,\ldots,x_k,y,h_1(x_1,\ldots,x_k,y),\ldots,h_n(x_1,\ldots,x_k,y))$$

is real recursive whenever h is of the class C^1 on the largest interval containing 0 in which a unique solution exists.

3. *Infinite limits*: if f is a real recursive vector with n $(k+1)$-ary components, then the vectors h, h', h'' with n k-ary components $(1 \leqslant i \leqslant n)$

$$h_i(x_1,\ldots,x_k) = \lim_{y\to\infty} f_i(x_1,\ldots,x_k,y),$$
$$h'_i(x_1,\ldots,x_k) = \liminf_{y\to\infty} f_i(x_1,\ldots,x_k,y),$$
$$h''_i(x_1,\ldots,x_k) = \limsup_{y\to\infty} f_i(x_1,\ldots,x_k,y),$$

are real recursive in the domain containing these points, where these limits exist for all $1 \leqslant i \leqslant n$.

4. Arbitrary real recursive vectors can be defined by assembling scalar real recursive components.
5. If f is a real recursive vector, then each of its components is a real recursive scalar.

From the physical point of view with such a definition we are ready to use only a finite amount of energy. The possibility of operations on undefined functions is excluded here: our functions are strict in the sense that for undefined arguments they are also undefined. But to obtain some interesting functions we should improve the power of this system by the use of the operators of infinite limits. Let us point out that introducing of infinite limits gives discontinuous functions. We should also remember that in some cases we can use limits in some real point. This is possible by transforming them into infinite limits. For example, $\lim_{y\to\frac{\pi}{2}} \sin xy$ can be written as $\lim_{y\to\infty} \sin x(\arctan y)$.

To illustrate further this transformation let us point out that if f is a $(n+1)$-ary real recursive function, then its derivative $\partial_y f(x_1,\ldots,x_n,y) = \lim_{\omega\to\infty}\omega(f(x_1,\ldots,x_n,y+\frac{1}{\omega}) - f(x_1,\ldots,x_n,y))$ is a real recursive function, whenever such a limit exists. For example, if we take $\lambda y\cdot 1/y$ then $\lim_{\omega\to\infty}(1/(y+\frac{1}{\omega}) - 1/y)\omega = \lim_{\omega\to\infty}\omega(y-y-\frac{1}{\omega})/[(y+\frac{1}{\omega})y] = -1/y^2$ is a real recursive function. Derivatives are physically realizable: the class of differential algebraic functions is closed under derivatives, making a large class of derivatives physically realizable.

2. Properties of analog computation by means of real recursive functions

Now we can consider a new problem. Are there different levels of difficulty in a computation if it goes in the analog way? The natural measure of a function difficulty can be joined with the degree of (dis)continuity. The above considerations lead us to the conception of the η-hierarchy which describes the level of nesting limits in the definition of a given function.

Syntactic descriptions of real recursive vectors are needed to formulate a precise definition. Some kind of symbols called basics descriptors are introduced for all basic real recursive functions. The combination of such descriptions for given real recursive functions will form a new description of another function. For basic functions we can propose: i^j_k is a k-ary description for projection I^j_k for all $1 \leqslant j \leqslant k$; $1_k, \bar{1}_k, 0_k$ are k-ary descriptions for constants 1, -1, 0 used with k variables. We must also add operator symbols (descriptors) for all introduced operators: d*r*–for a differential recursion, *c*—for a composition, *l*, *ls*, *li*—for a respective kind of limits (lim, limsup, liminf).

Definition 5 [18]. The collection of descriptors of real recursive vectors is inductively defined as follows:

- i^j_n is a n-ary description of I^j_n, $1 \leqslant j \leqslant n \in N$; 1_n is a n-ary description of $f(x_1,\ldots,x_n) = 1$; $\bar{1}_n$ is a n-ary description of $f(x_1,\ldots,x_n) = -1$; 0_n is a n-ary description of $f(x_1,\ldots,x_n) = 0$; for all $(x_1,\ldots,x_n) \in R^n$, $n \in N$;
- if $\langle h\rangle = \langle h_1,\ldots,h_m\rangle$ is a k-ary description of the real recursive vector h and $\langle g\rangle = \langle g_1,\ldots,g_k\rangle$ is a n-ary description of the real recursive vector g, then $c(\langle h\rangle,\langle g\rangle)$ is a n-ary description of the composition of h and g;

- if $\langle h\rangle = \langle h_1,\ldots,h_n\rangle$ is a k-ary description of the real recursive vector h and $\langle g\rangle = \langle g_1,\ldots,g_n\rangle$ is a $(k+n+1)$-ary description of the real recursive vector g, then $\mathrm{d}r(\langle h\rangle,\langle g\rangle)$ is a $(k+1)$-ary description of the function defined by differential recursion;
- if $\langle h\rangle = \langle h_1,\ldots,h_m\rangle$ is a $(n+1)$-ary description of the real recursive vector h, then $l(\langle h\rangle)$, $li(\langle h\rangle)$, $ls(\langle h\rangle)$ is a n-ary description of an appropriate infinite limit (respectively lim, liminf, limsup) of h;
- if $\langle f_1\rangle,\ldots,\langle f_m\rangle$ are n-ary descriptions of real recursive k-ary scalar functions $f_1,\ldots,f_m$, then $v(\langle f_1\rangle,\ldots,\langle f_m\rangle)$ is a k-ary description of the real recursive vector $f=(f_1,\ldots,f_m)$.

At the moment we can define the η-number for a description of some real recursive function f.

Definition 6 [18]. For a given n-ary description s of a vector function f let $E_i^k(s)$ (the η-number with respect to ith variable of the k-component) be defined as follows:

1. $E_i^1(0_n) = E_i^1(1_n) = E_i^1(\bar{1}_n) = 0$;
2. $E_i^m(c(\langle h\rangle,\langle g\rangle)) = \max_{1\leqslant j\leqslant k}(E_j^m(\langle h\rangle) + E_i^j(\langle g_j\rangle))$, where h is a n components k-ary vector and g is a k-components m-ary vector;
3. for a differential recursion we distinguish two cases:
   - $i \leqslant k$:

     $$E_i^j(\mathrm{d}r(\langle f\rangle,\langle g\rangle)) = \max(E_i^1(\langle f_1\rangle),\ldots,E_i^1(\langle f_n\rangle),E_i^1(\langle g_1\rangle),\ldots,E_i^1(\langle g_n\rangle),\ E_{k+1}^1(\langle g_1\rangle),\ldots,E_{k+1}^1(\langle g_n\rangle)).$$

   - $i = k+1$:

     $$E_i^j(\mathrm{d}r(\langle f\rangle,\langle g\rangle)) = \max_{0\leqslant m\leqslant n}(\max(E_{k+m+1}^1(\langle g_1\rangle),\ldots,E_{k+m+1}^1(\langle g_n\rangle))),$$

   where f is a n components k-ary vector and g is a n components $(k+n+1)$-ary vector;
4. $E_i^k(l(\langle h\rangle)) = E_i^k(li(\langle h\rangle)) = E_i^k(ls(\langle h\rangle)) = \max(E_i^k(\langle h\rangle),E_{n+1}^k(\langle h\rangle)) + 1$, where h is a k components $(n+1)$-ary vector.

The main idea of the above definition is to count nested limits in descriptions. In point (3) we should distinguish the case $i = k+1$ (differential recursion is given with respect to this variable); in this case $\langle f\rangle$ is not important for the counting.

For the n-ary description s of m components we can define now $E(\langle h\rangle) = \max_k \max_i E_i^k(\langle h\rangle)$ for $1\leqslant i\leqslant n$, $1\leqslant k\leqslant m$. Now we can deal with the η-number for a real recursive function.

Definition 7 [18]. For a given real recursive function f, let $\eta(f)$ be defined as the minimum of $E(\langle f\rangle)$ for all possible descriptions of the function f.

We are ready to conclude with the definition of the η-hierarchy as a family of $H_j = \{f : \eta(f)\leqslant j\}$. It will be comfortable to think about the η-hierarchy as a measure of the difficulty of real recursive functions. If $f\in H_j$, then j nested limits is used to define f. However we can patch functions defined by infinite limits, so j can be seen as the number of nested (non-parallel) η needed to patch the function f to the total function. Hence, as another equivalent definition we can suggest the following: if f is a real recursive function, then $E(f) = j$ if at most j nested η operations are necessary to create f_{total} such that f_{total} is everywhere defined and if $f(\bar{x}_0)$ is defined, then $f_{\mathrm{total}}(\bar{x}_0) = f(\bar{x}_0)$.

Let us illustrate the above part of the text with some examples.

Example 8. The functions $+$, $\times$, $-$, exp, sin, cos, $\lambda x\cdot\frac{1}{x}$, $/$, ln, $\lambda xy\cdot x^y$ are real recursive functions from H_0. The Kronecker δ-function, the signum function, absolute value, the Heaviside function Θ, the binary maximum max, the square-wave function s and the floor function are in H_1.

Let us give the examples of some functions which are significant in mathematics and can be expressed in terms of real recursiveness.

Example 9. The Bessel functions of the first kind J_v of order v (integer) are real recursive functions of the class H_0. The Euler's Γ-function and the Riemann ζ-function are real recursive functions from the class H_1.

Let us comment briefly Euler function: $\Gamma(x) = \int_0^\infty t^{x-1}\exp(-t)\,\mathrm{d}t$. It is simple to observe $\Gamma(x) = \lim_{s'\to\infty}\int_0^{s'} s^{x-1}\exp(-s)\,\mathrm{d}s$. Because $s^{x-1}\exp(-s)$ is a real recursive function and $\int_0^{s'} s^{x-1}\exp(-s)\,\mathrm{d}s$ is in H_0 hence Γ is in H_1.

Proposition 10 [18]. *The class of real recursive functions is a proper superset of the class of GPAC-computable functions.*

The above proposition is obvious considering our results that Γ Euler function and ζ Riemann function are real recursive functions and the result of Marion Pour-El [23] that these functions are not GPAC-computable.

For the proper analysis of functions it is important to control the domain and singularities of functions. We can postulate new operators which may check the points: are they in the domain of some functions or not.

Definition 11 [18]. For any function $f : \mathbb{R}^{n+1} \to \mathbb{R}$ let

$$\eta_y f(\bar{x}, y) = \begin{cases} 1 & \text{if } \lim\limits_{y\to\infty} f(\bar{x}, y) \text{ exists,} \\ 0 & \text{otherwise.} \end{cases}$$

The definitions of η_y^i and η_y^s are given by a replacement of lim by liminf and limsup, respectively, in the above equation.

Defined in this way $\eta_y f(\bar{x}, y)$ is a characteristic function for the set of such $\bar{x}$ that $\lim_{y\to\infty} f(\bar{x}, y)$ is well defined[1] (without singularities). Analogously, $\eta_y^i f(\bar{x}, y)$, $\eta_y^s f(\bar{x}, y)$ play the same role, respectively, for $\liminf_{y\to\infty} f(\bar{x}, y)$, $\limsup_{y\to\infty} f(\bar{x}, y)$. The problem arises whether such operators are real recursive operators. If the answer to this question is 'yes', we may patch any partial function to a total one. For example, let the function f be total and $F_{\text{total}}(\bar{x}) = \lim_{y\to\infty}(\eta_y f(\bar{x}, y))f(\bar{x}, y)$, $F(\bar{x}) = \lim_{y\to\infty} f(\bar{x}, y)$. The function $F_{\text{total}}(\bar{x})$ is total and has such a property that if $F(\bar{x})$ is defined, then $F_{\text{total}}(\bar{x}) = F(\bar{x})$. For points which are not in the domain of F we have $F_{\text{total}}(\bar{x}) = 0$.

Proposition 12 [18]. *The functions $\eta_y g$, $\eta_y^i g$, $\eta_y^s g$ are total real recursive functions if g is a total real recursive function.*

Now we can turn to some application of the η-operator. We consider a possibility of a process of Turing machines simulation by real recursive functions.

Let us consider a Turing machine given by the following description. It consists of an infinite tape for storing the input, output, and scratch working, and a finite set of internal states. All elements on a tape are strings. Without any loss of generality, we can choose some alphabet for these strings, the binary alphabet is a practical choice. The machine works in steps. At one step it scans the symbol from the current position of the tape (under the head of the machine), changes this symbol according to the current state of the machine and moves the position of the tape to the left or right with a transformation of state. Some states are distinguished as final, when the machine reaches one of them, then it stops. Our Turing machine model obeys to the classical constraints: (a) input is finite and (b) output is finite, regardless of the computation length.

Proposition 13. *There are real recursive functions from the class H_1, which can simulate any Turing machine, i.e., for a machine M there exists a real recursive function $f_M : \mathbb{R}^2 \to \mathbb{R}^2$ such that for the initial tape and state encoded into (x, y) the nth iteration $f_M^n(x, y)$ gives the codes of the tape and state after n steps of Turing machine.*

It can be mentioned that the process of simulation is especially important for universal Turing machines. The results in this area proved last year (e.g., [24]), give us the interesting restrictions of the size of such machines (for example, there exists a universal Turing machine for 5 states and 5 symbols) which leads us to a significant simplicity of the constructed function. It is worth pointing out that f_M can be analytical (see [10]).

[1] Whenever we say that lim, limsup, liminf are defined, we want to say that they belong to $\mathbb{R}$.

Let us signal a few important questions concerning Turing machines. The first problem is known as the halting problem: does the machine M for input (x,y) reach the final state? There is not a natural recursive characteristic function of this problem. The method of simulation of Turing machines given above can resolve it in the simple way with real recursive functions.

Proposition 14 [18]. *For any Turing machine M, there exists a real recursive function which is the characteristic function of the halting problem for M.*

Proof. We can define $F'_M(x,y,z) = f_M^{\lfloor z \rfloor}(x,y)$, then let

$$H_M(x,y) = (\eta_z F'_M(x,y,z))A(\lim_{z\to\infty}(\eta_z F'_M(x,y,z)) \cdot F'_M(x,y,z)),$$

where $A(x,y) = 1$ if the state written in x is final (accepting), 0 otherwise. The function $A(x,y)$ can be defined as $\sum_{s\in F}\sum_{a=0}^{m-1} h_{(m+1)n}(x - s - na)$. The function H_M is a real recursive characteristic function of the halting problem for the machine M. □

To obtain the function computed by M, it is enough to iterate the steps up to reaching the final state by the machine. If the machine M ends in the final state for some tape (x,y), then there exists such $n_0 \in N$ that the sequence $f_M^n(x,y)$ is constant for $n \geqslant n_0$. We can define the function $F_M(x,y)$ computable by M as

$$F_M(x,y) = \lim_{z\to\infty}[f_M^{\lfloor z \rfloor}(x,y) \times g(A(\lim_{z\to\infty}(\eta_z F'_M(x,y,z)) \cdot F'_M(x,y,z)))],$$

where g is a function not defined at 0, otherwise it takes value 1 (for example given as $\lim_{y\to\infty} \frac{1}{1-\exp(-|x|y)}$). Then F_M is defined whenever lim exists and the value of the function A is 1 (i.e., the Turing machine M reaches for the initial tape (x,y) the final state), otherwise is undefined.

Let us turn for a moment into the problems of computation beyond the power of Turing machines. The problem of infinity, which can appear in the sequel of not finishing computation, introduced problems into the computability theory and practice. The first step to improve this situation is directed to change the behavior of a Turing machine. For this purpose we may use an accelerated Turing machine [5]. Its description is the same as for a standard Turing machine, but a temporal pattern of steps is given. Each subsequent step is performed in half the time of the step before. Such machines could complete an infinity of steps in two time units only. This feature of accelerated Turing machines gives us the power to puzzle out the halting problem by programming the following algorithm: mark the first square on the tape by 0, change it only in the final (last) step to 1, if after 2 time units we have 0 in the distinguished square, then machine does not halt, otherwise it halts. However, some difficulties arise also in this model. Let us imagine the machine changing value of one square from 1 to 0 and, conversely, in all steps using only one non-final internal state. We can hesitate what is on the tape after all steps (in infinity) because in this case the computation diverges. The accelerated Turing machine can be simulated in the same way as the standard Turing machine with only one modification: in the definition of $F_M(x,y)$ it is not necessary to have the result (z_x, z_y) with a final state i written in z_x. Hence, the convergent infinite computations and finite computations both give the correct result, however the divergent computations have an undefined result. As we can observe, real recursive functions are quite a powerful tool of a description for this problem of infinite computation.

The above remarks prove that η-operator gives us an additional power to standard models of computation by controlling the domain of computable functions and machines. Such possibility is an effect of checking in a finite amount of time an infinite number of computation elements.

At this moment we can generalize our considerations from the notion of Turing machine up to a wider characterization of computability. We will proceed with the relations of natural numbers taken from the arithmetical hierarchy. The class $\Sigma_0^0 = \Pi_0^0$ contains only such relations which have recursive characteristic functions, i.e., which can be computed by Turing machines. The upper stages of this hierarchy can be constructed from the lower ones in the following way:

$$\Sigma_{n+1}^0 = \{P : (\exists P' \in \Pi_n^0)P(\bar{m}) \equiv \exists s P'(\bar{m}, s)\},$$
$$\Pi_{n+1}^0 = \{P : (\exists P' \in \Sigma_n^0)P(\bar{m}) \equiv \forall s P'(\bar{m}, s)\},$$

where $P \subseteq N^k$, $P' \subseteq N^{k+1}$, $k \geqslant 1$. To complete our hierarchies we can add the following equation $\Delta_n^0 = \Sigma_n^0 \cap \Pi_n^0$, $n \geqslant 0$.

The importance of the arithmetical hierarchy is connected with many fields. It can be observed as a kind of formal description of definability (see [21]). Its classes can be used to classify a complexity of mathematical notions (e.g., the definition of a limit of sequences is of Π_3^0 class). From the computability theory point of view we can see the arithmetical hierarchy as the levels of natural functions (given by their graphs) which are different in quantity of infinite 'while' loops necessary to their computation. Also linguistic problems of computer science can be expressed in terms of this hierarchy. The most known example is the one of the classes of recursive (Σ_0^0) and recursive enumerable (Σ_1^0) languages. In some sense we can also see this hierarchy as a measure of uncomputability of functions (or undecidability of relations). Especially, the halting problem is known to be Σ_1^0.

We are interested in a correlation of this infinite hierarchy of sets and relations to the η-hierarchy. Going to infinity with n for the function f_M^n and using the fact that all recursive sets and relations have Turing computable total characteristics, we get the following conclusion.

Corollary 15 [18]. *Every recursive set or relation (with argument from $\mathbb{N}$) is in H_2, i.e., $\Sigma_0^0 = \Pi_0^0 \subset H_2$.*

For the rest of the arithmetical hierarchy we have the following result.

Theorem 16 [18]. *The sets and relations from Σ_i^0, Π_i^0 belong to H_{i+2} for $i \geqslant 0$.*

This fact does not mean that a better real recursive restriction for the arithmetical hierarchy does not exist. Let us analyze one aspect of the analytical hierarchy. This hierarchy exceeds strongly the arithmetical one. We can deal with especially important class Π_1^1. Let us mention that Π_1^1 sets can be uniformized by sets of the same class. Moreover, for any $n \in \mathbb{N}$ the classes Σ_n^0 and Π_n^0 are subsets of Π_1^1. The class of Π_1^1 relations is defined by a function quantifier used on an arithmetical relation: $R \in \mathbb{N}^{k+1}$ is Π_1^1 if $R(\bar{x}, y) \equiv (\forall f : \mathbb{N} \to \mathbb{N})Q(\bar{x}, f(y))$, where Q is from some level of the arithmetical hierarchy.

Proposition 17. *Any relation $R \in \Pi_1^1$ is in H_6.*

The levels of the analytical hierarchy and their relation to the η-hierarchy can be analyzed with the μ-operator like in [13]. Because the μ-operator may be replaced by infinite limits (see [15]) the remaining part of the analytical hierarchy can be obtained in this way.

3. Real functions computable by Turing machines

Let us change our approach. From now on we will use only classical computational devices like Turing machine. To define real functions computable by such machines we have to consider a way of an approximation of real numbers by natural ones. This method is well known (e.g., [9]), here we use an approach such as in [34].

Let us present the method of a connection of real computations with Turing machines. The first step is devoted to some coding of rational number (from $\mathbb{Q}$) into natural numbers. We use Cantor pairing function $\langle i,j\rangle = \frac{(i+j)(i+j+1)+j}{2}$ and inverses $\pi_1(\langle i,j\rangle) = i$, $\pi_2(\langle i,j\rangle) = j$. Of course, we can extend this for more arguments: $\langle i,j,k\rangle = \langle i, \langle j,k\rangle\rangle$. It is possible to define an effective numbering $v_{\mathbb{Q}} : \mathbb{N} \to \mathbb{Q}$ such that $v_{\mathbb{Q}}(n) = \frac{i-j}{k+1}$, for $n = \langle i,j,k\rangle$. A function $f : \mathbb{N}^k \to \mathbb{Q}$ is called computable iff $f(n_1, \ldots, n_k) = v_{\mathbb{Q}}(g(n_1, \ldots, n_k))$ for some function $g : \mathbb{N}^k \to \mathbb{N}$ computable by Turing machine.

The next step is connected with computable sequences of real numbers. A sequence $(x_n)_{n\in\mathbb{N}}$ of real numbers is computable in this sense if there is computable (in the sense of the previous paragraph) double sequence $(r_{n,m})_{n,m\in\mathbb{N}}$ of rational numbers which converges uniformly and effectively to $(x_n)_{n\in\mathbb{N}}$, namely there is a Turing computable function $e : \mathbb{N}^2 \to N$ such that

$$(\forall n, m, k \in \mathbb{N})(k \geqslant e(n,m)) \to |r_{n,k} - x_n| < 2^{-m}.$$

Let us explain that the above definition can be expressed in such an informal way: the sequence of real numbers is computable if there exists such a Turing machine that for the given index n of some element of this sequence and for the given precision 2^{-m} this machine computes the code of such rational number r, which differs with x_n less than 2^{-m}.

At the last step we would like to define a real function, which can be approximated by Turing machine. We want to have the following intuition preserved—if a given rational argument is in the known neighborhood of x then the Turing machine gives rational number in the known neighborhood of $f(x)$. Moreover, we should have a possibility to decide by some parameter how small is the neighborhood.

So we would like to have for a function f an effective procedure which uses a Turing machine M_f: for a given precision n and for a given sequence $r_{m,k}$ (produced by another Turing machine) convergent to x, this procedure finds such m_0, k_0 that $M_f(r_{m_0,k_0}) = q$ and $|q - f(x)| < 2^{-n}$.

With this intuition we can give the definition of computable real functions, here in the sense of functions approximable by Turing machines.

Definition 18 [34]. A real function $f : [a, b] \to \mathbb{R}$ is called computable, if f maps every computable sequence $(x_n)_{n\in\mathbb{N}}$ of real numbers to a computable sequence $(f(x_n))_{n\in\mathbb{N}}$ and there exists a recursive function $d : \mathbb{N} \to \mathbb{N}$ such that

$$(\forall x, y \in [a, b])(\forall n \in \mathbb{N})|x - y| < 2^{-d(n)} \to |f(x) - f(y)| < 2^{-n}.$$

Such definition gives us quite different look at the problem of computability over the reals. Here we restrict the set of real functions in such a manner, that we consider only functions, which can be approximated effectively by Turing machines. In this sense they are physically computable by digital computers, because we can always find as the result a rational number near to the original value of a function with any desirable precision. Such functions cannot be treated as any instance of hypercomputation. Let us point out that e, π are real computable, that means we have a method to a rational number in any neighborhood of them; Chaitin's Ω is not real computable in this sense.

4. GPAC and H_0 versus Turing machines

We introduce a simple concept—by an analogy with the recursive functions of Kleene, whenever a function is defined only with composition and differential recursion ($f \in H_0$), we call f a primitive real recursive function.[2]

Proposition 19 [18]. *Every primitive real recursive function f defined on the closed domain $D \in R^n$ is GPAC-computable.*

However, let us point out that there are functions (like $\lambda x \cdot |x|$ in the interval $[-1, 1]$), which are bounded with their derivatives but they, or some of their derivatives, are not continuous (and not primitive real recursive ones).

We can observe that the model of analog computation given by real recursive functions includes GPAC-computable functions.

Theorem 20 [18]. *Every GPAC-computable function with real recursive numbers as parameters is a real recursive function.*

Now let us do the first step to analyze when some functions generated by analog computation are beyond the Turing limits of computations. Of course, real computable functions described in the previous section are under this limit. But what is the situation of GPAC-computable functions? This question is additionally important because, as we can observe from the above propositions, GPAC-computable functions form the basic level of analog computations, which is a common part of EAC-computable functions and real recursive functions.

The answer can be found in Rubel's paper [26], where the following theorem (in the slightly different formulation) is proved.

[2] There is a slight difference since (classical) primitive recursive functions are always total and primitive real recursive functions can be partial.

Proposition 21 [26]. *For any GPAC with a locally unique solution, which is analytical on an open interval I containing 0 with an analytical output f, if all the constants are rational numbers, then there is an open subinterval $0 \in J \subset I$, on which f is digitally computable.*

In [26] the phrase 'f is digitally computable' means that there is a uniform algorithm for producing polynomials p_n with rational coefficients such that $|p_n(x) - f(x)| < \frac{1}{n}$ for all $x \in J$. It is simple to observe that this condition can be transformed into conditions satisfied by real computable functions in the sense of Definition 18.

Now, we can consider another important case of analog computation. Let us look closer at the primitive real recursive functions, i.e., at the class H_0. All functions from this class are GPAC-computable by Proposition 19. From these results we have a corollary that all primitive real recursive functions are real computable. We can also obtain the same result by the following observation. The basic functions (constants and projections) are real computable. Composition of two real computable functions is also real computable. Hence, the only one not obvious case is a definition by differential recursion. However, an integration used by this operator can be approximated by some kind of numerical integration with a control of a precision, which gives us a real computable solution. So, we have the corollary.

Corollary 22. *Every primitive real recursive function can be approximated by Turing machines.*

The above results teach us that analog computation in some cases does not increase the computational power beyond Turing border. This is not sufficient (as it is sometimes suggested) to use real numbers to cross restrictions of Turing computability. Of course, the precise comparison of Turing and analog computability depends on a detailed analysis of all operations allowed in the discussed models.

5. What can us take beyond the Turing limit?

Let us try to enumerate some examples of hypercomputational properties of analog computation and to find the common elements.

Moore in his paper [13] proves the theorem that the halting problem is solvable by $\mathbb{R}$-recursive functions if we allow to use the zero-finding μ-operator. However, we can avoid the use of this, somehow unnatural for the analog world of mathematical analysis, operator. The good choice seems to be guaranteed by infinite limits.

Theorem 23 [15]. *If $f(\bar{x}, y) : \mathbb{R}^{n+1} \to \mathbb{R}$ is a real recursive function then the function $g : \mathbb{R}^n \to \mathbb{R}$, $g(\bar{x}) = \mu_y f(\bar{x}, y)$ is real recursive too.*

Hence, we obtain a corollary that the halting problem can be solved by real recursive functions, when we use infinite limits in the solution. The same fact, by a different method, is given in our Proposition 14.

Let us discuss briefly another analog device. It is an open problem whether EAC can be simulated by Turing machines (in general, by digital computers). However, this is a known fact that EAC is stronger that GPAC and it can compute some difficult functions (e.g., Γ-function, ζ-function). We can observe that these properties are connected with infinite limits, namely the proofs of EAC-computability for such functions use the mentioned operation. It would suggest that the power of EAC is strongly connected with limits incorporated in its structure.

In the classical framework we can create uncomputable functions (relations) extending the set of Turing computable functions to the whole arithmetical hierarchy. As a consequence of the definition of this hierarchy we find unrestricted quantifiers as tools leading toward uncomputability. Hence, we need to analyze the method of quantifiers usage.

For every function $f : R^{n+1} \to R$ we can construct such a real recursive function $\rho_f : R^n \to R$ that

$$\rho_f(\bar{x}) = \begin{cases} 1 & \exists y \in N f(\bar{x}, y) = 0, \\ 0 & \forall y \in N f(\bar{x}, y) \neq 0. \end{cases}$$

To this effect we start with a description of the function $f_c(\bar{x}, y) = 1 - \delta(f(\bar{x}, y))$. This function has the following property $f_c(\bar{x}, y) = 1 \equiv f(\bar{x}, y) \neq 0$, $f_c(\bar{x}, y) = 0 \equiv f(\bar{x}, y) = 0$. It is easy to observe that now

$$\lim_{z\to\infty}\prod_{j=0}^{z} f_c(\bar{x},j)=\begin{cases}0 & \exists y\in Nf(\bar{x},y)=0,\\ 1 & \forall y\in Nf(\bar{x},y)\neq 0.\end{cases}$$

Hence $\rho_f(\bar{x})=1-\lim_{z\to\infty}\prod_{j=0}^{\lfloor z\rfloor} f_c(\bar{x},j)$. We should indicate two points. The first: real recursive functions are closed under the product operation. It can be defined as an iteration of the function $t_f\colon R^{n+2}\to R^{n+2}$, $t_f(\bar{x},y,i)=(\bar{x},yf(\bar{x},i),i+1)$, hence

$$\prod_{i=0}^{n} f(\bar{x},i)=I_3^2(t_f^n(\bar{x},1,0)).$$

The second: let us analyze the stage of η-hierarchy which contains $\rho_f(\bar{x})$ if $f\in H_i$, where $i\in N$ is a given number. The function f_c is in H_{i+1} and consequently by properties of an iteration $\prod_{j=0}^{n} f_c(\bar{x},j)\in H_{i+1}$. Finally, we can claim that $\rho_f\in H_{i+2}$.

The same method is applicable to the general quantifier. In this way we can, once more, conclude that an uncomputable character of natural functions and relations given by quantifiers appear in the context which can be explained by real recursive functions with infinite limits.

We can go back for a moment to accelerated Turing machines. It is a well-known fact that such machines could solve the halting problem. But, as we explained in the previous section, this possibility arises from the property of reaching infinite number of steps in finite time—this means that hypercomputable properties of accelerated Turing machines flow from infinite limits, too.

In the light of the above considerations hypercomputation for analog devices has a strong connection with infinite limits. The use of infinite limits gives a possibility of an analysis for infinite number of cases in finite amount of resources (time, space, energy). Therefore, at this stage of our work we can present the uncontroversial thesis: with infinite limits some hypercomputational properties are added to analog models of computation.

6. Are limits physically possible?

From the previous considerations we can deduce that for establishing the boundaries of practically realized models of computability, the nature of the material world becomes essential. It is important, however, to become aware of an obvious fact that we do not possess a direct knowledge of this quality of the Universe. That is why an analysis of its features and limits always takes place by means of physical theories. These theories become the only way to perceive quantitative relations that occur in the physical world.

Therefore, we face the next boundary of our analysis. We cannot discuss ultimate boundaries of computability but the limits of computability possibilities that result from a physical theory which is regarded as given. However, it may appear that such a far-reaching claim, namely the postulate of realizing infinity (energy, time) in some finite sector of physical reality is not acceptable to every physical theory.

The fact that physical reality models does not exhibit super-Turing capabilities is claimed, for example, by Penrose. In [22] he stresses this point before he talks about the (non-computable) ultimate physical theory and the human mind: *Now, where do we stand with regard to computability in classical theory? It is reasonable to guess that, with general relativity, the situation is not significantly different from that of special relativity—over and above the differences in causality and determinism that I have just been presenting. Where the future behavior of the physical system is determined from initial data, then this future behavior would seem* (*by similar reasoning to that I presented in the case of Newtonian theory*) *also to be computably determined by that data* (*apart from unhelpful type of non-computability encountered by Pour-El and Richards for the wave equation, as considered above—and which does not occur for smoothly varying data*). *Indeed, it is hard to see that in any of the physical theories that I have been discussing so far there can be any significant "non-computable" elements.*

But it occurs, though, that the above assumptions are not obvious. For more rigor we can restrict the considerations at least to commonly approved (not particularly exotic) physical theories. Two examples of physical theories which can be seen as allowing hypercomputability will be presented at this moment.

The first is Newtonian mechanics. In 19th century P. Painlevé together with H. Poincaré proposed a particular analysis of a problem connected with the mechanics of heavenly bodies, that is the question of n-bodies.

In this system of n-bodies a solution of an equation system of movements for n gravitational interacting bodies is sought. P. Painlevé and H. Poincaré opened a discussion not about the way to discover a particular solution, but about an analysis of qualities of these solutions. There is a crucial question whether there may exist such solutions that contain a singularity. It is obvious that a situation of this kind happens when two, from all the described by the problem, bodies collide. Yet, the question arises whether the singularity may appear without any collision. The answer to this question was given by Xia [33] in 1992. He claimed that for a problem of five bodies in the three-dimensional space there exist non-collision solutions. The Xia answer causes throwing one of the bodies to infinity in finite time. As it can be seen, Newtonian mechanics allows finite realizations of infinity and potentially supports a possibility of calculations exceeding the limits of the Turing machine.

Hence, we have reasons to believe that the n-body dynamics has hypercomputation capabilities and it is possible to explore them. Xia's paper [33] showing that an infinite number of mechanical events can happen in finite time opens a way of thought. There is an open way to translate the halting problem into the n-body problem in classical mechanics: it is necessary to show that the subset of initial data that go off to infinity in finite time codes a universal machine. In Tipler's book [32], he conjectured that universal initial data exists and it seems that the universal initial data is of measure zero in the space of all initial data.

An obvious aim that appears at this moment is relating similar considerations to the physical theories that are regarded as currently valid (not as Newtonian mechanics). For this purpose we will use the theory of general relativity. There are such solutions of Einstein's equations in which there exist a time-like half-curve γ as well as a point p in spacetime such that the entire stretch γ is contained in the chronological past of p. Such spacetime structures (i.e., anti-de Sitter spacetimes) have been examined by physics with pointing out possible material systems that fulfill the required qualities (compare [8]). Moreover, the descriptions of the usage of such systems to create computing systems [28] have been proposed. Summing up, this theory, allows conducting an infinite number of operations in finite time of an appropriate observer.

From the point of view of analog computation, time is very important ingredient of such systems. Hence, the next element of this discussion will be devoted to a properties of time. Is time a continuous object or rather a discrete one? In our analog models it seems to be natural way of thinking, that variables (at least one of them) change in a continuous way in the time dimension of the Universe. This point is under discussion among experts. Some of them reject the continuous nature of time, others (see for example Newton-Smith in [20]) claim that on empirical grounds, continuous time is a better model for our Universe than discrete one.

The above results do not entitle us to accept the thesis that hypercomputability is possible in our world. They show, however, that the possibility of crossing the Turing machine barriers is, in the light of some physical theories, real.

As we have observed, a new cognitive situation is introduced. The boundaries of computability become valid only for a stated physical theory. Moreover, they receive provisional and temporary character. When a physical theory regarded as the proper description of the Universe changes, there may occur a change in computability boundaries. The theory of computability gained additionally a relative character, this time in relation towards the physical theory assumed as a starting point in a construction of computing systems.

7. Inner restrictions of analog computation with limits

The previous parts of the paper can be seen as a claim that with a use of analog paradigm in computation equipped with limits we create a framework where all problems are solvable. Such result would be, of course, disappointing. An omnipotent system of computation would lose its constructive and algorithmic character. If all analytical functions (or moreover set-theoretical functions) would be computable, then a real meaning of computation would disappear.

Because the system of real recursive functions was presented as very general, let us concentrate on it. Hence, here by undecidable problems we understand such problems which cannot be solved by real recursive functions. It may be observed, from the below results, that many important problems remain undecidable in this sense.

Proposition 24 [17]. *The problem of domain for real recursive functions is undecidable.*

Proposition 25 [17]. *The problem of identity of two real recursive functions given by their descriptions* (*coded into natural numbers*) *is undecidable by a real recursive function.*

We will say that a description is elegant if it has the property that no smaller description exists. We should then look at the size of descriptions, and, since they are written in characters, we can size them in quite different ways. As with Chaitin's elegant *LISP* programs (e.g., [4]) we cannot prove that a description is elegant.

Let us assume some computable linear orders into the set of description numbers (i.e., into $\mathbb{N}$). The examples of such an order can be: $m \prec n$—the description number m codes a shorter description, or with same length but occurring lexicographically before the description of code n; $m \prec_l n$—the order is lexicographical among the set of descriptions; $m \prec_s n$—we compare the number of limits involved in the descriptions of codes m and n, then the number of differential recursions, then the number of compositions, in the case that all numbers are equal we use the length, and later the lexicographical order. All mentioned linear orders are computable, i.e., they can be decided by classical recursive functions, hence by real recursive functions too.

Proposition 26 [17]. *The problem of minimal code is undecidable by real recursive functions.*

We give a corollary of the above proposition. In the case of the order $\prec_s$ some function M counts the minimal number of limits needed to define a function given by some description. Since M is not real recursive, we can conclude that the problem of establishing the minimal level of the η-hierarchy (no matter the number of differential recursions and compositions) is not decidable in our framework.

On the other side of such problems, we can analyze computability of some numbers. A real recursive number x is a real recursive function with arity zero (or an equivalent formulation states that there exists a unary real recursive function $f : \mathbb{R} \to \mathbb{R}$ with the property $f(0) = x$). The examples of real recursive numbers are the integers, but also the rationals, and the algebraic numbers (see [16] for a full proof). Transcendental numbers such as $\pi = 4\arctan(1)$ where $\arctan(0) = 0$, $\partial y \arctan(x) = \frac{1}{1+x^2}$, Napier's $e = \exp(1)$, Euler's $\gamma = -\lim_{z\to\infty} \int_0^z e^{-x} \ln x \, dx$, but also Chaitin's halting probability Ω are real recursive in different levels of the limit hierarchy. However, their number is countable. As analog quantities, these numbers are given once and for all as entire quantities (see [13]) and not as decimal expansions like in the case of the usual definition with Turing machines.

A Rice-like general undecidability and incompleteness theorem for real-defined and real-valued functions—appear in the paper by da Costa and Doria [7], which was submitted for publication in October 1990 and commented by Stewart in Nature (see [31]) in August 1991. Similar results appear in the Ph.D. thesis of Moore ([11, p. 119]) which was submitted to the Math Department in Cornell University in January 1991; those results had been obtained before December 1990.

Herein we proceed to show how we can adapt the same results (their classical presentation can be found in [6]), namely Rice's theorem, to the set of real recursive functions. We start with the real recursive variant of another classical result.

Proposition 27 (The analog s–m–n theorem [17]). *For each $m, n \geqslant 1$, there is the total real recursive $(m+1)$-ary function s_n^m such that for integer numbers $x_1, \ldots, x_m$: $\phi_e^{m+n}(x_1, \ldots, x_m, y_1, \ldots, y_n) = \phi^n_{s_n^m(e,x_1,\ldots,x_m)}(y_1, \ldots, y_n)$.*

This leads us to the well-known and important Rice's theorem in the context of analog computation.

Proposition 28 (The analog Rice's theorem [17]). *Suppose that $\mathscr{B}$ is non-empty set of real recursive functions not coinciding with the whole class. Then the problem λx. "$\phi_x \in \mathscr{B}$" is undecidable.*

In other words, any property that real recursive functions can have, like being injective, or onto, or having an infinite domain, or having an infinite range, or being total, is undecidable unless it is trivial (corresponding to the empty set and to the set of all real recursive functions). Now, in terms of dynamical systems, these questions concern basins of attraction. These basins are in general non-real recursive, i.e., there is no hyper-algorithm that will tell us whether or not a point is in them. In fact, we can even recall Theorem 10 from [12], which is proved by analog Rice's theorem.

Proposition 29 (Moore's undecidability theorem [17]). *The following questions about continuous-time dynamic systems are undecidable.*

1. *Given a point x and an open set A, will x fall into A*?
2. *Given a point x and a periodic point p, will x converge to p*? *Will a dense set of points converge to p*?
3. *Is the set of periodic points on a given cylinder infinite*? *Dense*?

We think of a conceptual programming language that can be interpreted partially as analog circuits of a kind pioneered by Bush in 1930, and extended and speculated later by Rubel working on Shannon's model of Bush's Differential Analyzer. This programming language make use of limits, a mathematical concept that is not in general physically realizable. Thus (invoking informally the physical *CT* Thesis) such computational model is beyond the Turing limit, although functions remain constructible and countable in number. The hypercomputational character of the model has, however, *nothing to do with the fact it handles real numbers* and real numbers (as in [30]) can code for non-computable functions. The hypercomputational capabilities are associated with the operators, especially with the taking of limits.

How powerful it is, this model suffers from the same classical limitations as the standard Turing model of computation: the classical results of computability theory can be parameterized to our programming language to show that, e.g., the halting problem of analog computation is not solvable. This is quite different from [30], since although real (but non-computable) numbers are incorporated in their model (as net weights), neural nets with real weights are not countable and thus the same diagonalization methods of classical computability can not be applied to a set of non-countable nets. Herein, we have the *same s–m–n theorem* and the *same Rice's theorem* as in classical computability but with different interpretations.

8. Conclusions

Let us summarize the article with a few remarks concerning the relation between analog computation and hypercomputation.

First, the use of real numbers does not lead automatically to a possibility of computations beyond Turing borders. We could observe that although GPAC is an analog device it can only find solutions to such problems which can be solved (approximately but with any desirable precision) by Turing machines [26] too.

Next, the study of analog computation does not have to be inspired by hypercomputational motivation. There are good reasons to see analog computation as a useful tool of mathematics to study problems of classical computability (with already obtained results in such fields as Grzegorczyk's hierarchy [3,2]), complexity theory [19] and arithmetical hierarchy [18]. Moreover, there are some results which can compare analog computation with fields of mathematics (e.g., descriptive set theory [14]). Hence, the results for various models of analog computation should not be treated as the direct support for hypercomputation.

However, it could be admitted that on some levels of analog computations (in the η-hierarchy [18] or, probably, for EAC-computable functions [27]) we can find classically uncomputable functions. Let us point out that some tools for descriptions of such functions are also given in classical computability theory. Namely, the upper levels of the arithmetical hierarchy or degree theory consists of uncomputable, in Turing sense, functions [21]. So, it would be suggested that such wide and rich theories of analog computation do not speak about real possibility of computation, but form degrees of uncomputability by finding and classifying main uncomputable operators and structures. We believe that the current results give the strong support for the thesis of an existence of natural connection between uncomputable character of some function and infinite limits involved (explicitly or implicitly) in its definition.

Finally, we think that an important aspect of the discussions of hypercomputation is connected with physics. Hence, the question about possibility of some computation beyond Turing limit is the question about necessary conditions for physical theories describing our Universe. In this aspect analog computation has some importance, as the theory closer in its formulation to physical theories than classical computability theory.

References

[1] L. Blum, F. Cucker, M. Shub, S. Smale, Complexity and Real Computation, Springer, 1998.
[2] O. Bournez, E. Hainry, An analog characterization of elementarily computable functions over the real numbers, in: Proceedings of Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Lecture Notes in Computer Science, vol. 3144, Springer-Verlag, 2004, pp. 269–280.

[3] M.L. Campagnolo, C. Moore, J.F. Costa, An analog characterization of the Grzegorczyk hierarchy, J. Complexity 18 (4) (2002) 977–1000.
[4] G.J. Chaitin, The Limits of Mathematics, Springer, 1998.
[5] B.J. Copeland, Even Turing machines can compute uncomputable functions, in: C.S. Calude, J. Casti, M.J. Dinneen (Eds.), Unconventional Models of Computation, Springer-Verlag, 1998.
[6] N.J. Cutland, Computability: An Introduction to Recursive Function Theory, Cambridge University Press, 1980.
[7] N.C.A. da Costa, F.A. Doria, Undecidability and incompleteness in classical mechanics, Int. J. Theor. Phys. 30 (8) (1991) 1041–1073.
[8] G. Etesi, I. Nemeti, Non-Turing computations via Malament–Hogarth spacetimes, Int. J. Theor. Phys. 18 (2002) 341–370.
[9] A. Grzegorczyk, On the definition of computable real continuous functions, Fund. Math. 44 (1957) 61–71.
[10] P. Koiran, C. Moore, Closed-form analytic maps in one or two dimensions can simulate Turing machines, Theor. Comput. Sci. 210 (1999) 217–223.
[11] C. Moore, Complexity in dynamics systems, Ph.D. Dissertation, Cornell University, 1991.
[12] C. Moore, Generalized shifts: unpredictability and undecidability in dynamical systems, Nonlinearity 4 (1991) 199–230.
[13] C. Moore, Recursion theory on the reals and continuous-time computation, Theor. Comput. Sci. 162 (1996) 23–44.
[14] J. Mycka, Real recursive functions and Baire classes, Fund. Inform. 65 (3) (2005) 263–278.
[15] J. Mycka, μ-Recursion and infinite limits, Theor. Comput. Sci. 302 (2003) 123–133.
[16] J. Mycka, J.F. Costa, The computational of continuous dynamic systems, in: Machines, Computations, and Universality: 4th International Conference, MCU 2004, Saint Petersburg, Russia, Lecture Notes in Computer Science, vol. 3354, 2005, pp. 164–175.
[17] J. Mycka, J.F. Costa, Undecidability in continuous dynamic systems, Logic J. IGPL, in press.
[18] J. Mycka, J.F. Costa, Real recursive functions and their hierarchy, J. Complexity 20 (6) (2004) 835–857.
[19] J. Mycka, J.F. Costa, The $P \neq NP$ conjecture in the context of real and complex analysis. J. Complexity, in press.
[20] W. Newton-Smith, The Structure of Time, Routledge & Kegan Books, 1984.
[21] P. Odifreddi, Classical Recursion Theory, Elsevier, 1989.
[22] R. Penrose, The Emperor's New Mind, Oxford University Press, 1989.
[23] M.B. Pour-El, Abstract computability and its relation to the general purpose analog computer, Trans. Am. Math. Soc. 199 (1974) 1–28.
[24] Y. Rogozhin, Small universal Turing machines. Universal machines and computations, Theor. Comput. Sci. 168 (2) (1996) 215–240.
[25] L.A. Rubel, Some mathematical limitations of the General-Purpose Analog Computer, Adv. Appl. Math. 9 (1988) 22–34.
[26] L.A. Rubel, Digital simulation of analog computation and Church's thesis, J. Symb. Logic 54 (3) (1989) 1011–1017.
[27] L.A. Rubel, The extended analog computer, Adv. Appl. Math. 14 (1993) 39–50.
[28] O. Shagrir, I. Pitowsky, Physical hypercomputation and the Church–Turing thesis, Minds Mach. 13 (2003) 87–101.
[29] C. Shannon, Mathematical theory of the differential analyser, J. Math. Phys. MIT 20 (1941) 337–354.
[30] H.T. Siegelmann, E.D. Sontag, Analog computation via neural networks, Theor. Comput. Sci. 131 (1994) 331–360.
[31] I. Stewart, Deciding the undecidable, Nature 352 (1991) 664–665.
[32] F. Tipler, The physics of immortality: modern cosmology, God and the resurrection of the dead, Anchor (1997).
[33] Z. Xia, The existence of noncollision singularities in newtonian systems, Ann. Math. 135 (3) (1992) 411–468.
[34] X. Zheng, K. Weihrauch, The arithmetical hierarchy of real numbers, Math. Logic Quart. 47 (1) (2001) 51–65.